wordgraph Documentation

Release 0.0.1

Tennessee Leeuwenburg

September 01, 2014

Contents

CHAPTER 1

Why

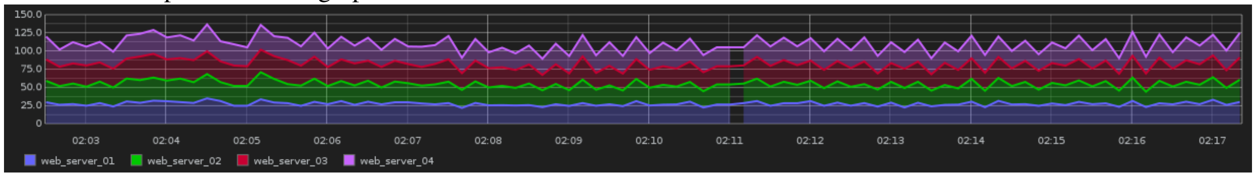
This project is intended to support anyone who is doing screen reading for any reason. This could be for vision-impaired people, or just people who like to listen to graphs while jogging, or just to get a handle on what's going on. Could potentially be used for generic data description also.

Kate Cunningham gave an amazing keynote at PyCon AU 2014 regarding accessibility of web pages. Someone came up with the idea that graphs were a problem, and that maybe it would be possible to come up with a language description for those who wanted to understand the information, but couldn't see the graph.

CHAPTER 2

What

When the data points for this graph:



Are passed through *wordgraph*, it generates the following description:

> “This graph shows the relationship between time and metric. The x axis, time, ranges from 04 Aug 2014 02:05:10 to 04 Aug 2014 02:20:00. The y axis, metric, ranges from 0 to 37.3. It contains 4 series.”
>
> —Wordgraph, 5 August 2014 (some obvious room for improvement)

CHAPTER 3

Who

Maintainer: Tennessee Leeuwenburg @tleeuwenburg

Code Contributors (in order of first contribution):

1. Tennessee Leeuwenburg
2. Aaron Iles
3. Nick Farrel
4. Ryan Stuart
5. Rebecca Dengate

CHAPTER 4

Basic Usage

A core library takes an abstract graph description and produces English-language text.

The basic usage is:

```
> import wordgraph
> text = wordgraph.describe(data, source_type)
```

CHAPTER 5

Contents

5.1 Integration Guide for Other Systems

This document outlines how people who are maintaining other systems, such as the graphite web application, or any other back-end system for depicting graphs, or custom-authored pages integrating graphs, can best interface with and use wordgraph.

5.1.1 Overview

The text generated by wordgraph can be integrated in two ways: during page generation on the server, or potentially as a web API for client-side graph generation from javascript. There is no permanent, public web service for wordgraph. (That would be awesome! If you want to contribute server resources, let me know! Better yet – help set it up too!). The "beaten path" (insofar as a two-day-old product can support) is to integrate the text generation directly into the server-side web application which is generating the HTML for your pages.

The text produced can be dropped into the ALT text for the html image elements, providing screen readers with a way to find the text without presenting it to visual users. Alternatively, the text could be placed inside a text element underneath the graph if you wish to present the text to all web users.

The use of the package is not technically restricted to web applications, but GUI framework integration is beyond the experience of the package authors and so is not addressed in this guide.

The basic approach is to install the wordgraph package into your application, then call into its top-level method as follows. More detailed instructions for supported packages will be provided in following sections of this document. The text that is returned can be customised in a number of ways. These options are addressed in the API documentation as they do not directly affect system integration.

```
> import wordgraph
> text = wordgraph.describe(data, source_type)
```

5.1.2 Graphite

This system is designed for potential inclusion into the graphite software package. The source data for the original tests and input cases for wordgraph comes from the JSON data which is supplied by that system when its graphs are generated. Unfortunatey, a full integration walkthrough is not available at this stage. A simple example walkthrough would be an extremely useful community contribution. Calling wordgraph with the JSON data and the source type of 'graphite' is all that is required.

```
> import wordgraph
> text = wordgraph.describe(data, source_type='graphite')
```

5.1.3 Matplotlib

Matplotlib support is currently stalled, and not yet complete. The goal is to support this through direct introspection of the "figure" objects produced by matplotlib. Many users of matplotlib will be using either the 'pylab' or 'pyplot' / 'plt' interface for generating their graphs. These implicitly create a "currently active" figure which can be retrieved using "plt.gcf()". Users creating figures by hand can pass them in individually.

```
> import wordgraph
> text = wordgraph.describe(figure, source_type='matplotlib')
```

It will be difficult to fully support complex plots, in particular it will be challenging to support figures which consist of multiple sub-plots. Until the package is substantially further expanded, please pass these in individually.

5.1.4 Custom or one-off graphs

Custom or one-off graphs can be supported in a number of different ways. If you have a time-series graph and the raw data, the simplest approach is most likely to re-create the graphite standard data structure and utilise the graphite interface.

If you wish to create a more complex graph description, you could consider using an alternative API entry point. It is not especially complicated to create the underlying data dictionaries which contain the describable values expected by the language generation subsystems. You could consider directly interfacing with the wordgraph "realisers". Please look at the API docs and the tests directory for additional information on how to go about this approach.

5.1.5 Extensions and Other Systems

The package maintainers would love to support a wider variety of systems to help support the community. However, it is important to realise that there is no money made from this activity, and those who are supporting it are doing so on a strictly volunteer basis. What this means is that sometimes it may take some time to respond to complex emails, and the extent of support available will be highly variable depending on the other commitments of the individuals involved. If you would like to read more about how open source development works, search for "Open Source development" and there will be a wealth of information returned to you.

There is a lot that can make it easier to support integration of a new system. Obviously, "pull requests" (code contributions) are a direct way of improving the functionality of the system. However, the following contributions will all be gratefully received:

1. Stories about how you are using the system, who this helps, and the impacts made.
2. Input data examples from other systems
3. Detailed descriptions of requirements for improvements to any aspect of the system
4. Example graphs and expected text
5. Defect or issue reports where something didn't meet expectations

The package maintainers will do their best to support additional use cases and requests as best possible.

5.2 Developer Guide

5.2.1 Getting Set Up

This package is currently Python 2/3 compatible and we will attempt to maintain that, in order to minimise the barrier to adoption and increase the potential reach of the library. However, Python 3 is the lingua franca, with Python

2.7 supported through backporting. Please attempt to minimise incorporating additional libraries which do not have Python 2.7 support, and all included libraries must have Python 3 support.

There are not very many external dependencies, however getting fully set up has been surprisingly challenging for some specific environments. For this reason, a functional Vagrant configuration has been supplied to support development within a virtual machine environment.

For those wanting to develop "natively", the package can be successfully installed on recent builds of Fedora, Ubuntu and OSX, and probably older builds also. You will need to set up a Python 3 environment. We recommend using a virtual environment in order to minimise any potential conflicts with other system packages.

5.2.2 Detailed Architectural Overview

The package is divided into the following key areas:

+The top-level 'wordgraph' method are contained within 'describer.py', which contains logic for passing data and requests off to the appropriate handlers given the source and type of the incoming data. +The "analysers" file, which contain methods for recognising describable patterns within individual data series within a graph. +The "grapher" file, which contains methods for creating describable-graph data structures from specific types of source input data. Individual Graph classes can be created which will make assumptions about the data being passed in. +The "realiser" sub-module, which contains the code for generating language given the describable features of the graph identified in the grapher module.

5.2.3 Choosing Work Tasks

If you prefer, please feel free to just start working in your own direction. No guarantee is made that pull requests will be integrated. If you want to contribute back to the repository, it is best to contact the package maintainers to discuss the direction of your work before investing too much time.

A list of useful tasks are included in the github issues list. If you start working on an issue, even if you are simply looking into it and haven't yet got a solution in mind, please leave a message there. If nothing else, it will give the package maintainers some increased visibility of interest in that particular area. You may also be able to contribute background information and context that could help someone later working on that task.

The documentation pages also often contain suggestions for future work which haven't made it to the issues list, but are still potentially rewarding areas. Please feel free to add these to the issues list, and if you address them, please also update the documentation to match.

5.2.4 Developer Workflow

The approach of this package is fairly open and encouraging of direct contribution. Please don't commit any code which breaks test compliance, although additional input cases can be added as "xfailing" prior to the development of a solution. For those of you with the 'commit bit', please get a code review (in person or via a pull request) for anything more than about 20 lines. Feel free to get a code review on one-line changes if you would like!

The initial development team was formed during the PyCon AU 2014 sprints, and there is no specific developer induction process at this stage. For now, just flag your interest. It is likely that the process of gaining commit access will involve the submission of a few good-quality pull requests, but please don't be hesitant if you are considering getting involved.

There is also a heavy emphasis on good documentation and test practises. The goal of the project is to support those who want to get involved. There is an "all welcome" approach, not a "rocket scientists only" one.

5.3 API

5.3.1 Wordgraph Top Level

The high level interface for *Wordgraph* is the `wordgraph.describe` function.

`wordgraph.`**`describe`**`(`*data*, *source=None*, *language='English'*, *demographic='summary'*`)`

Describe the supplied graph object, together with a hint about the source of that object.

@return: None if there was no description text generated for the graph @return:

Supported sources include: – Raw data (graphite-style)

Unsupported sources include: – graphite full integration – matplotlib (under dev) – dot, networkx? – json, text?

5.3.2 Grapher Module

This module contains the code for extracting the values needed for text generation out of the supplied data package.

The Graph class defines the basic interface, which is simply to be able to get a dictionary of descriptor variables back. It is extended by AutoGraph which is also abstract, and defines the interface for automated data ingestion.

AutoGraph is extended by GraphiteGraph, which does something. This Graph class makes various non-general assupmtions about the incoming data, consistent with the data being generated by the "graphite" web graphing application.

Useful extensions to this module would include a broader selection of specific graph types, supporting a wider array of input data use cases.

class `wordgraph.grapher.`**`AutoGraph`**

Base class for automated data ingestion types. Defines the call to data ingestion.

`auto_ingest``(`*raw_data*`)`

Perform the processing on raw data and prepare the dictionary of values needed for the realisers

class `wordgraph.grapher.`**`Graph`**

Base class for all other graph types, defining the data access method which will be used by the Realiser classes to fetch descriptor data.

`as_dict``()`

Return a dictionary of values for use by the realiser classes.

class `wordgraph.grapher.`**`GraphiteGraph`**

Expects data as produced by the "graphite" web application.

`as_dict``()`

`auto_ingest``(`*raw_data*`)`

Stores the raw data into self.raw_data Stores structured data into self.processed_data Creates the response for as_dist ad self.result

class `wordgraph.grapher.`**`MPLGraph`**

`as_dict``()`

`auto_ingest``(`*fig*`)`

`wordgraph.grapher.`**`generic`**`()`

5.3.3 Analysers Module

Analysers are used to process a single series, and produce structured output describing the series.

As a user of this module, you will usually simply invoke get_analysis(points).

Analysers do not care about things like axis labels; they only need to find the best way of representing the data in the graph.

The analysers' results are language agnostic, and will be translated into natural language elsewhere.

To add a new analyser, simply subclass the FixedIntervalAnalyser and implement the two methods. get_validity() will be used to asses which analyser is most suitable for describing the data, while get_result() does the actual analysis. Then add it to the list of analysers.

Note that the values passed into the analysers are just the y-values; you should not need to know the x-values, and you can assume the x distance between consecutive points is constant.

class `wordgraph.analysers.`**`FixedIntervalAnalyser`**`(`*points*`)`

Given a series of y-values, associated with fixed x-axis increments, provide analysis of it.

`get_result``()`

Returns a (jsonable) representation of the values using this analyser.

`get_validity``()`

Returns a float representing how well this analyser can describe these values. 1 represents a perfect fit, 0 represents no fit.

class `wordgraph.analysers.`**`LinearDistribution`**`(`*points*`)`

`get_result``()`

`get_validity``()`

`name` **= 'linear'**

class `wordgraph.analysers.`**`NormalDistribution`**`(`*points*`)`

`generate_idealised_normal``()`

Using the original x values of the input data, return a new set of points which as based on the assumed mean and stddev

`get_result``()`

`get_validity``()`

(hacky) approach: Assuming that we do have a normal distribution. Then compare the theoretical cumulative normal distribution to the actual distribution at various points, and base the overall score on the overall deviation.

We will assume the X values start at 0 and increment by 1 for each point in the series.

`name` **= 'normal'**

`total_size`

`x_value_at``(`*proportion*`)`

What is the x value which places this proportion of the graph to the left of this place?

class `wordgraph.analysers.`**`RandomDistribution`**`(`*points*`)`

No meaninful pattern in the data.

`get_result``()`

get_validity()

name = 'random'

wordgraph.analysers.**assert_fixed_interval**(*points*)

wordgraph.analysers.**get_analysis**(*points*)

wordgraph.analysers.**get_best_analyser**(*values*)

Instantiate a bunch of analysers and return the one which suits this data best.

wordgraph.analysers.**phi**(*x*)

Cumulative distribution function for the standard normal distribution.

Note – this expects the standard deviation normalisation to have already been done outside of this fn.

Taken from the python math docs. Using to avoid a scipy dependency for now – scipy is very hard to install via pip due to O/S package dependencies. We want to support a simple install process if at all possible.

5.3.4 Language Realisation API

Python Module Index